Réalisation d'une application de conversion de température.

version 1

Interface Homme-Machine : Android

Voici les objectifs de ce sujet :

— Comprendre l'IDE `Android Studio`.
— Créer une activité.
— Connaître les widgets de base disponibles.
— Maîtriser l'ajout de réactivité.
— Récupérer les widgets par programmation.
— Réaliser des calculs simples pour afficher les résultats.
— Comprendre les mécaniques de traçage ("Logs") pour pister son application.
— Ajouter des retours haptiques pour informer l'utilisateur.

Description générale de l'application

Nous verrons les premiers éléments du développement sur Android au travers d'une application permettant de convertir des températures *Celsius* en *Fahrenheit* (et vice-versa) dans un premier temps, puis, dans un second temps, nous ajouterons les *Kelvin*. Les aspects calculatoires ne nous intéressent pas et nous insistons sur la compréhension générale des composants graphiques (`widgets`) et leur manipulation pour obtenir une application fonctionnelle de ce point de vue. Nous attacherons aussi une importance non négligeable à l'ergonomie de l'application et à l'intuition de son fonctionnement avec les outils que nous présenterons tout au long des séances.

1 Création de votre projet sous `Android Studio`

La première étape est la création d'un projet. Nous supposons que vous avez installé `Android Studio` correctement et qu'au moins un SDK (≥ 23) est présent. [[Rédiger une procédure à part expliquant comment récupérer Android et gérer les SDK. Vérifier si on conserve la version 23 minimale.]]

Une fois l'IDE lancé, lancez la procédure de création d'un projet (menu *File* → *New* → *New Project...*), vous devez obtenir la fenêtre suivante :

Cette fenêtre permet de choisir la première interface de votre application, ainsi que le code associé simplifiant des étapes redondantes. Pour nos premières séances, nous choisirons toujours l'onglet *"Phone and Tablet"* et une "*activité vide*" (*"Empty Activity"*) afin de ne pas être perturbé et mieux comprendre les étapes de développement. Une fois ce choix réalisé, passez à l'étape suivante en bas de la boîte de dialogue, pour obtenir la fenêtre suivante :

Cette étape consiste à spécifier le paramétrage global de votre application en précisant :

— Le nom de votre projet (attention, certains caractère ne sont pas supportés en fonction de votre système de fichier en raison de la création d'un répertoire portant ce nom : il est conseillé d'utiliser des caractères non acentués, et aucun espace).
— Le nom du paquet (au sens `Java`) où l'ensemble de vos classes seront affectées par défaut.
— La localisation de votre projet sur votre disque dur (vous pouvez bien sûr modifier ce paramètre à votre gré).
— Le langage de développement. Actuellement, Google soutient le langage `Kotlin` mais conserve encore la compatibilité avec `Java` (**contrainte du cours** et nous vous conseillons `Java`).
— Le SDK minimal indique la version minimale d'Android où votre application pourra s'exécuter.
— [[Mettre à jour la phrase qui suit.]] En 2020, le cours impose une version ≥ 23. Vous pouvez adapter ce paramètre à votre configuration mais influence énormément de paramétrage interne aussi bien en terme de développement ou de sécurité. [[Pour simplifier le choix aux étudiants, et puisque tu demandes une version ≥ 23, autant fixer la version à 23 pour tout le monde, non ? Cela diminue les chances d'avoir des questions plus tard relatives à des versions toutes différentes.]]
— [[Mentionner de NE PAS cocher le "Use loegacy android support libraries" ?]]

A présent, vous pouvez terminer la création de votre projet. Félicitations !

2 Comprendre l'interface `Android Studio`

2.1 Interface globale

L'interface `Android Studio` est complète (bien qu'elle soit un peu lourde pour certaines configurations) et fournit beaucoup de boutons qu'il faut connaître ou qui s'utilisent à des moments précis. L'objectif est de connaître les plus importants :

Commençons avec l'explorateur de projet qui regroupe l'ensemble des fichiers dans le projet en rouge sur la figure ci-dessus. Les fichiers sont multiples et cela inclut entre autres les fichiers de développement, les fichiers de configuration et/ou les fichiers générés. D'ailleurs, vérifiez que la mention pointée par la flèche rose indique bien le filtrage `Android` pour masquer les fichiers inutiles. La zone verte permet de configurer le projet et surtout l'environnement `Android` en terme de SDK disponible et de fonctionnalités spécifiques ; il permet aussi de créer et gérer l'ensemble de vos émulateurs. Vous verrez à l'usage d'autres éléments. Voici les fichiers principaux présents dans un projet :

- `app` : représente la racine de votre application.
- `manifests` : contient le fichier de configuration de votre application (`AndroidManifest.xml`). Ce fichier donne les permissions requises par votre application, l'ensemble des interfaces, *etc.* Dans la mesure du possible, vous éviterez de modifier le fichier à la main.
- `java` : contient l'ensemble des sources de votre projet. Par défaut, vous avez 3 paquets : le vôtre et deux de tests que nous ne verrons pas immédiatement. Ils servent à tester des applications.
- `res` : est l'ensemble des ressources `XML` qui définissent le paramétrage des votre application, réparties en sous-répertoires : `drawable` contient les images, `layout` l'ensemble des interfaces, `mipmap` les ressources multi-résolution et `values` des ressources valuées. Ce répertoire est accessible par programmation en passant par la classe statique `R` qui est auto-générée et auto-synchronisée lors des constructions de votre application.
- `Gradle script` : regroupe l'ensemble des scripts de reconstruction de votre projet et donc indirectement des informations sur la configuration des dépendances de votre projet. Il est très fortement conseillé de *ne pas modifier ces fichiers* sauf indications contraire et avec parcimonie.

2.2 Concepteur d'interface

Intéressons-nous maintenant à la conception d'une interface graphique. Pour cela, cherchez dans l'explorateur de projet le fichier `activity_main.xml` présent dans le répertoire `layout`, vous pouvez vous aider de la capture d'écran suivante :

Le concepteur d'interface manipule et interprète le fichier `XML` sous-jacent, ce qui explique que vous n'avez pas à modifier manuellement le fichier (sauf cas très très exceptionnel). Le concepteur est découpé en plusieurs bloc que nous pouvons réduire de la façon suivante :

Nous présentons rapidement les différents éléments et il est très important que vous les manipuliez au cours des séances afin d'acquérir le plus d'expérience et d'aisance dans le développement `Android` :

- La zone rouge regroupe l'ensemble des contrôles graphiques (boutons, listes, cases à cocher, ...) mis à votre disposition pour réaliser vos interfaces ; ils sont regroupés en catégories.
- La zone orange fournit une visualisation des composants présents dans votre interface et leur ordre au sein du fichier `XML`. Elle affiche les dépendances pour jouer avec les conteneurs (c'est-à-dire les composants capables de contenir d'autres composants).
- La zone bleue permet de configurer la prévisualisation de votre interface en modifiant la densité de pixels, la résolution, la version du SDK, le thème, la langue, ...
- La zone verte fournit des outils pour faciliter le placement de certains composants graphiques.
- La zone rose décrit l'ensemble des propriétés pour le contrôle sélectionné.
- Le reste représente la zone de conception où vous pouvez placer les composants, prévisualiser l'interface, fixer des contraintes de placement...Elle se sépare en deux : l'interface en couleur réelle et une version dite `blueprint` qui permet de faire ressortir les bordures des objets pour mieux apprécier le placement.

Vous vous habituerez très vite à manipuler cette fenêtre, qui est une partie très importante de la conception d'une application.

3 Réalisation du convertisseur de température

3.1 Conception de l'interface

En utilisant l'IDE, on vous demande de réaliser l'interface de notre convertisseur de température entre les *Celsius* et les *Fahrenheit* uniquement. Pour cela, vous allez placer les composants graphiques suivants :

— `TextView` définit un texte devant s'afficher sans possibilité de modification par les utilisateurs. Il permet entre autres de servir d'étiquette pour expliquer un champ de saisie et de donner un résultat.

— Les champs de saisie de valeur numérique (`Number` dans l'IDE) autorisent l'utilisateur à préciser une valeur (ici numérique). Vous utiliserez ce champ pour affecter la température en entrée.

— `RadioButton` représente un bouton de sélection. Il doit s'intégrer un `RadioGroup` (ce dernier est qualifié de *conteneur*), ce qui permet de choisir un `RadioButton` parmi l'ensemble contenu. Vous utiliserez ce contrôle pour choisir l'unité de votre saisie (à savoir *Celsius* ou *Fahrenheit*).

— `Button` insère un bouton cliquable par l'utilisateur pour lancer un calcul, ici la conversion souhaitée.

Une fois que vous avez identifié tous ces contrôles dans le concepteur, il faut les placer dans votre interface. Premièrement, vous devez supprimer le texte `Hello World` ajouté par défaut par `Android Studio`. [[Si les étudiants font comme moi, ils vont voir un pictogramme d'avertissement comme quoi le texte dans la widget est "hard coded". Tu expliques ce type de problème à la fin de cette section, dans l'encadré en vert, mais peut-être faudrait-il y faire référence dès maintenant, ou bien écrire de ne pas s'inquiéter pour le moment et d'attendre les explications de l'encadré qui viennent plus tard.]]

NB : ATTENTION pour Xavier : je me suis aperçu que dans mon jargon j'utilisais conteneur et layout indifféremment. Mais en Android il y a une légère différence entre les deux. Je ne sais pas si c'est utile de rentrer à ce détail, mais il faudra sans doute harmoniser le discours
[[J'ai bien lu la remarque. On peut essayer de conserver "layout" dès qu'on parle du conteneur spécifique pour la disposition des éléments et "conteneur" pour tous les autres types de regroupements de widgets.]]

Pour placer vos contrôles graphiques, vous devez comprendre la notion de conteneur graphique [["gestionnaire de disposition" ? plutôt que "conteneur graphique" ? C'est un peu lourd mais on évite le terme "conteneur" et dans la suite, on utilisera uniquement "layout"]] (ou de `layout` en anglais). Ces composants servent à la mise en page automatique de votre interface. Il existe plusieurs types de conteneurs où chacun permet de placer les composants suivant une politique précise, nous pouvons citer notamment :

— `LinearLayout` : place les composants les uns à la suite des autres (verticalement ou horizontalement).

— `GridLayout` : place les composants dans une grille.

— `ConstraintLayout` : donne des contraintes de positionnement des composants selon des points d'ancrage. Ce sont les 4 poignées autour du `TextView` que vous pouviez voir sur la figure présentant le détail du concepteur. C'est ce `layout` qui est recommandé par Google mais vous pouvez le changer selon vos besoins.

Le `ConstraintLayout` impose au développeur de mentionner au moins une contrainte verticale (les poignées haut ou bas) et au moins une contrainte horizontale (les poignées gauche et droite). Il est également important de comprendre l'ordre des connexions et la façon à laquelle vous exprimez les contraintes auront un impact visuel sur votre interface finale. En bref, il n'est pas simple de le maîtriser parfaitement il faudra souvent s'y reprendre à plusieurs fois pour arriver à un résultat satisfaisant. Aussi, il est conseillé dans un premier temps de ne pas vouloir faire quelque chose de "parfait" mais de se contenter du suffisant, avec l'expérience et la pratique vous pourrez mieux apprécier les difficultés (et pensez à appeler votre enseignant pour essayer de vous aider).

Essayez de créer votre interface pour produire quelque chose de similaire à l'image de gauche ci-après, vous pouvez bien sûr varier le placement, ou votre style à votre gré. La figure à droite affiche les contraintes utilisées pour arriver à une interface satisfaisante :

Lorsque vous aurez terminé, vous devez manipuler les outils de prévisualisation pour visualiser les différences de comportement de votre interface, en particulier la rotation portrait/paysage pour apprécier le `ConstraintLayout` et toutes vos contraintes en action en rester dans la zone d'affichage. D'ailleur,s vos contraintes doivent permettre de conserver l'ensemble des contrôles dans la zone d'affichage sinon vous devez modifier vos contraintes.

Pour Xavier : est-ce qu'on parle ici du layout landscape variant ? Je pense qu'au vu de l'interface il est préférable que non.
[[D'accord avec toi, pas la peine d'en parler ici]] [[Mise à jour 2021-09-13 : en fait, on demande à mettre en place la variante "Landscape" dès la première séance de TA. Donc soit on écrit quelques lignes à ce propos dans ce fichier, soit on en parle directement en TP.]]

Attention

Dans la zone de hiérarchie, vous pouvez avoir des pictogrammes d'alertes ou d'erreurs qui sont évalués à chaque modification de votre interface. Il est important de les lire au moins une fois. Pour cela, vous devez cliquer sur l'un des pictogrammes et une zone de message apparaît avec le détail du pictogramme comme sur la figure ci-dessous. Notez que certains messages proposent un ou plusieurs boutons `"Fix"` qui peuvent résoudre le problème (ce qui n'empêche pas d'essayer de les résoudre par vous-même d'abord).

Lorsque l'on clique sur le bouton présent dans l'avertissement *"Codé en dur"* (*"Hardcoded text"*), l'outil suggère d'extraire la valeur textuelle.L'image ci-dessous illustre ce cas et propose d'écrire le texte de l'étiquette dans un fichier de ressource contenant toutes les chaînes `strings.xml`. Il suffit alors de valider la boîte de dialogue et l'avertissement disparaît. Vous pouvez ensuite visualiser le contenu de `string.xml` pour repérer les modifications qu'il a reçues.
Au fur et à mesure de votre apprentissage, ce sont très souvent les mêmes contraintes qui reviennent et vous pourrez les gérer très facilement et mécaniquement.

3.2 Préparer votre application à l'International

Si vous avez suivi l'interface de démonstration (selon la volonté de l'auteur), cette dernière est en anglais. Avec `Android Studio`, il est très facile d'adapter son application pour différentes langues et de laisser `Android` choisir la langue la plus adéquate en fonction du périphérique et de l'utilisateur (on parle alors de "*locale*"). Dans la zone des outils de prévisualisation, trouvez le pictogramme indiquant la langue (`"Default (en-us)"`) et cliquez dessus pour ouvrir l'éditeur de traduction.

Vous verrez l'ensemble des textes extraits (1 phrase par ligne) dans votre application. Vous pouvez utiliser la colonne `"Untranslatable"` pour indiquer les éléments n'ayant pas de traduction, comme le nom de l'application par exemple. Ensuite, cherchez le bouton en forme de planète qui permet d'ajouter une locale et choisissez le français. Attention, le système de locale permet de différencier la langue française et la langue française spécifique à un pays : vous pouvez donc tout à fait proposer une traduction française et une spécifique pour les Canadiens par exemple.

Lorsque vous avez terminé, vous pouvez fermer la fenêtre et modifier les paramètres de la visualisation pour observer votre interface dans une autre langue. Attention, il est toujours très important de le faire au moins une fois, car les traductions changent les longueurs des textes, qui peuvent sortir de la zone visible ou ne pas s'agencer correctement. Ces éléments font partie du test d'interface.

Information

Félicitations, vous avez fini votre première interface. Elle est visuelle seulement mais si vous avez bien réalisé toutes les étapes, vous pouvez exécuter votre application pour visualiser votre interface dans un émulateur ou sur un smartphone.

3.3 Réalisation du code

Au vu des objectifs de l'application, il faut mettre en place le comportement suivant : lorsque l'utilisateur a saisi une température et fixé l'unité de cette température, il lui suffit de cliquer sur le bouton pour lancer le calcul de conversion et afficher le résultat. Pour cela, vous allez d'abord détecter les cliques sur le bouton pour lancer une fonction que vous définirez et qui réalisera la conversion.

3.4 Préparation

Vous travaillerez dans le fichier source principal : `MainActivity.java`. Premièrement, créez un ensemble d'attributs à notre classe :

```
public static String TAG = "UPConvTemp"; // Identifiant pour les messages de log

private EditText editInputTemp; // Boite de saisie de la température
private RadioButton rbCelsius; // Bouton radio indiquant si la saisie est en Celsius
private RadioButton rbFahrenheit; // Bouton radio indiquant si la saisie est en Fahrenheit
private TextView dispResult; // Le TextView qui affichera le résultat
```

Vous avez vu que l'interface graphique est un fichier `XML` alors que vous développez dans un langage de programmation. En IHM, on dit qu'`Android` "gonfle" (*inflate*) l'interface. Donc, pour récupérer l'objet `Java` de nos composants, le SDK fournit une fonction nommée `findViewById` qui permet de chercher l'objet `Java` à partir de l'identifiant déterminé dans le concepteur du composant (chercher la propriété `ID`). La récupération des objets se fait lorsque l'interface à fini de "gonfler", donc **à la fin de la fonction** `OnCreate`. Vous devez donc, pour chaque attribut de composant graphique vu précédemment, récupérer l'objet`Java` associé en vous inspirant de la ligne ci-dessous qui récupère la boîte de saisie à partir de son identifiant (bien évidemment, vous aurez sans doute un autre nom comme identifiant) :

```
editInputTemp = findViewById(R.id.editInputTemp);
```

3.5 Retour haptique

L'émergence des téléphones et leur spécificité en termes de puissance, taille d'écran, résolution, quantité de capteurs embarqués ..., induit de l'ingénierie pour communiquer une information à l'utilisateur autre qu'un message long à lire. Parmi les retours haptiques très répandus, nous pouvons citer des sons de notifications précis (pensez à la réception d'un SMS que nous reconnaissons sans même regardez notre téléphone) ou les mécanismes de vibration qui peuvent différer selon la situation ou pour confirmer un mouvement. Voici deux mécaniques très simples en `Android` que vous pouvez/devez utiliser tout au long de votre carrière.

Le code de la fonction ci-dessous exploite la vibration d'un téléphone que vous pouvez bien sûr, reprendre en l'état ou adapter à votre guise. La vibration nécessite une permission (regardez bien les messages d'avertissement, l'IDE vous proposera d'ajouter la bonne permission).

```
public void vibrate(long duration_ms) {
        Vibrator v = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
        if(duration_ms < 1)
                duration_ms = 1;
        if(v != null && v.hasVibrator()) {
                // Attention changement comportement avec API >= 26 (cf doc)
                if(Build.VERSION.SDK_INT >= 26) {
                        v.vibrate(VibrationEffect.createOneShot(duration_ms,
                                        VibrationEffect.DEFAULT_AMPLITUDE));
                }
                else {
                        v.vibrate(duration_ms);
                }
        }
        // sinon il n'y a pas de mécanisme de vibration
}
```

Une autre façon de communiquer une information éphémère à l'utilisateur est d'afficher un petit message très court à lire appelé un `toast` en Android. Le code suivant illustre une fonction permettant d'utiliser un `toast` :

```
public void toast(String msg) {
        Toast.makeText(this, msg,Toast.LENGTH_SHORT).show();
}
```

Vous avez deux types de retour que vous pouvez utiliser comme vous le souhaitez dans vos projets. Attention cependant aux abus, qui peuvent perturber la compréhension des utilisateurs d'une part et surtout, prenez soin de vous fixer des conventions claires dans votre démarche. Par exemple, deux vibrations avec des durées de vibration trop similaires rendant la compréhension ambiguë.

3.6 Fonction de conversion

Écrivez la fonction de conversion nommée `public void action_convert(android.view.View v)`. Cette fonction suppose que l'ensemble des attributs sont correctement connectés et vous devez réaliser le code décrit dans les étapes suivantes :

1. Extraire et convertir la valeur saisie par l'utilisateur (cette valeur est textuelle).
2. Identifier l'unité choisie en regardant quel bouton radio est coché.
3. Réaliser la formule adéquate de conversion.
4. Formater le résultat pour afficher la valeur convertie et l'unité cible (par exemple on demande à afficher 32°C).
5. Si une erreur survient, vous devez afficher un message de log cohérent et faire un retour d'erreur succinct à l'utilisateur.

Quelques informations supplémentaires pour vous aider à réaliser le code demandé :

— Pour retrouver la liste des attributs/méthodes des composants graphiques, vous avez une référence très complète sur le site : `https://developer.android.com/reference/packages`

— La conversion *Celsius* en *Fahrenheit* est : $F = (C * 9/5) + 32$

— La conversion *Fahrenheit* en *Celsius* est : $C = (F - 32) * 5/9$

— Vous devez lier le clique sur le bouton de cette fonction depuis le concepteur d'interface en sélectionnant le bouton et en cherchant la propriété `OnClick`. Si en dépliant la liste, vous ne voyez pas la fonction de conversion, cela signifie que vous n'avez pas respecté sa signature.

Vous pouvez lancer l'application et vérifier qu'elle fonctionne comme prévu. Félicitations à vous.